**Durée : 2h AUCUN DOCUMENT - AUCUNE CALCULATRICE**

**Toutes les réponses de cette partie sont à remplir directement sur le sujet. Indiquez ci-dessous votre nom, prénom et groupe.**

**NOM : ------------------------------------ PRENOM : ------------------------- GROUPE : ----------------**

**Question de cours (5 points)**

1. Quelle est la différence entre une classe et un objet ? 1 point

Classe = définition d’un nouveau type (exemple : classe Voiture)

Objet = instance d’une classe (exemple : objet « mercedes » est instance de la classe Voiture)

2) Qu'est-ce que le principe d'encapsulation ? 1 point

Les attributs sont en général privés (sauf cas exceptionnels)

On utilise les accesseurs : getters (pour récupérer la valeur d’un attribut privé d’un objet) et setters (pour modifier la valeur d’un attribut privé d’un objet)

3) A quoi sert le mot-clé *abstract* ? 1 point

A définir le prototype d’une méthode non implémentée dans une classe abstract

4) Quelle est l'intérêt d’un ArrayList par rapport à un tableau ? 1 point

Principal intérêt : tableau dynamique extensible à « l’infini »

Autres intérêts annexes : données contigües en mémoire, pas de « trou » dans le tableau, accès par indice via un getter

5) Que signifie le mot *protected* ? 1 point

Accessible (directement, sans besoin de getters et setters) par (toutes les méthodes de) la classe elle-même et toutes les classes descendantes (filles ou sous-classes OK aussi)

Des extraits de la documentation des API Java sont fournis en **annexeS** dans les deux dernières pages.

**Exercice 1 : Compréhension de code de base (2 points)**

public class Essai

{

public void swap(char tab[], int a, int b) {  
 char c = tab[a];  
 tab[a] = tab[b];  
 tab[b] = c;  
}  
  
public void display(char tab[]) {  
 for(int i = 0; i < tab.length; i++) {  
 System.out.print(tab[i]+",");  
 }  
}  
  
public void mystery(char tab[]) {  
 int min = 0;  
 for(int i = 0; i < tab.length-1; i++) {  
 for(int j = i; j < tab.length; j++) {  
 if (j == i) min = j;  
 if (tab[j] > tab[min]) min = j;  
 }  
 swap(tab, i, min);  
 }  
}  
  
public static void main(String argv[]) {  
 char tab [] = {'a', 'b', 'd', 'c'};

Essai test = new Essai();  
 test.mystery(tab);  
 test.display(tab);  
}

}

Ecrire la sortie du programme (ce qui sera affiché à l'écran lors de son exécution) 1 point. Justifiez ! 1 point

d, c, b, a,

Ce programme trie le tableau par ordre décroissant grâce à la méthode mystery, puis échange avec la méthode swap les 2 valeurs de cases d’indices i (indice de la valeur maximum) et min (indice de la valeur minimum des cases parcourues) du tableau

**Exercice 2 : (5 points)**

Soit la classe suivante dont le rôle et de remplir aléatoirement *n* entiers dans un *vecteur* de type **ArrayList**, d’afficher ses valeurs et de calculer la moyenne. La valeur de *n* sera fournie par l'utilisateur.

import java.util.ArrayList ;

public class Dummy  
{  
 private ArrayList<Integer> vecteur; // vecteur de type ArrayList d’objets de la classe Integer  
 public Dummy(){…} // constructeur

// méthodes  
 public void remplirAlea(int n){…}   
 public void afficher(){…}  
 public float moyenne(){…}

// main  
 public static void main(String argv[]){…}  
}

1) Implémenter le constructeur qui instancie le vecteur 1 point

vecteur = new ArrayList<Integer> () ;

2) Implémenter la méthode remplirAlea(int n) qui remplit aléatoirement entre 0 et 20 les *n* éléments du vecteur1 point

for(int i=0 ; i<n ; i++) {

Ramdom r = new Random();

int alea = r.nextInt(21) ;

vecteur.add(new Integer(alea)); // vecteur.add(alea) accepté aussi

}

3) Implémenter la méthode moyenne() qui retourner la moyenne du vecteur 1 point

float somme=0 ;

for(int i=0 ; i<vecteur.size() ; i++) {

somme = somme + vecteur.get(i).intValue();

}

return somme/vecteur.size();

4) Implémenter la méthode afficher() qui affiche les valeurs du vecteur 1 point

for(int i=0 ; i<vecteur.size() ; i++) {

Systsme.out.println(vecteur.get(i).intValue());

}

5) Ecrire les lignes du main qui saisit la valeur de *n* qui doit être positif (à blinder) et appellent les 3 méthodes ci-dessus. L’affichage de la moyenne se fera dans le main. 1 point

Dummy test = new Dummy() ;

Scanner clavier = new Scanner (System.in) ;

int n ;

do {

System.out.println("Entrez un nombre positif :") ;

n = clavier.nextInt() ;

} while (n <= 0) ;

test.remplirAlea(n);

System.out.println("Moyenne = " + test.moyenne());

test.afficher() ;

**Exercice 3 (5 points)**

1) Ecrire une classe *Personnage* comportant deux attributs entiers **protected** : *point\_vie* et *point\_defense*. Implémenter 2 constructeurs : le premier par défaut, le second avec tous les paramètres nécessaires pour les attributs. 1 point

public class Personnage {

protected in point\_vie, point\_defense ;

public Personnage() { point\_vie=0; point\_defense=0; }

public Personnage(int v, int d)) { point\_vie=v; point\_defense=d; }

}

2) Ecrire une classe *Magicien* héritant de *Personnage* ayant en plus un attribut entier **private** : *point\_magie*. Implémenter un constructeur qui hérite du second constructeur de *Personnage* et initialise son attribut *point\_magie* avec le paramètre. 1 point

public class Magicien extends Personnage {

private int point\_magie ;

public Magicien(int v, int d, int m) {

super (v, d);

point\_magie = m ;

}

}

3) Ecrire deux classes *MagicienBlanc* et *MagicienNoir* héritant de *Magicien*. Dans la classe *MagicienNoir*, implémenter un constructeur par copie d’un objet de *Magicien* en paramètre. 1 point

public class MagicienBlanc extends Magicien {

}

public class MagicienNoir extends Magicien {

public MagicienNoir(Magicien papa) {

this.point\_vie = papa.point\_vie ; // le this (de l’objet du MagicienNoir) est facultatif ici

this.point\_defense = papa.point\_defense;

**//-0.25 point pour this.point\_magie = papa.point\_magie; car point\_magie private**

}

}

4) Donner l'ordre d'appel des constructeurs quand je créé un objet de type *MagicienBlanc*. 1 point

Dans l’ordre, les constructeurs de *MagicienBlanc* puis *Magicien* puis *Personnage* (des descendants vers ceux de la classe mère)

5) Soit p1 un objet de type *Personnage* et mg1 un objet de type *MagicienNoir*. Peut-on faire l'instruction suivante : "mg1 = p1;" ? Expliquer brièvement pourquoi. 1 point

Non car l’objet p1 de la classe mère *Personnage* ne peut pas référencer l’objet mg1 de la classe fille *MagicienNoir* alors que le contraire p1 = mg1 ; est naturel.

**Exercice 4 - Héritage et polymorphisme (3 points)**

Considérons une classe mère *Vehicule* ainsi que trois classes filles *Moto*, *Camion* et *Voiture*.   
Toutes ces classes possèdent une méthode *afficher()*.  
On souhaite gérer un garage.

1) Ecrire la classe *Garage* avec un attribut *vehicules* contenant un ensemble d’objets de la classe *Vehicule*~~, ainsi qu'une méthode~~ *~~affiche~~r*. 1 point

public class Garage {

private Vehicule[] vehicules ; // private ArrayList<Vehicule> vehicules est très bien aussi ☺

}

2) Implémenter la méthode *afficher*  de la classe *Garage*. Cette méthode affichera les informations des véhicules, en utilisant la méthode *afficher* des classes filles déjà fournies 1 point

public void afficher() {

// version 1 : avec le tableau

for (int i=0 ; I < vehicules.length; i++)

vehicules[i].afficher();

// version 2 : avec ArrayList

for (int i=0 ; I < vehicules.size(); i++)

vehicules.get(i).afficher();

}

3) Que peut-on dire de la méthode *afficher* de la classe mère *Vehicule* et donc de la classe elle-même ? 1 point

Cette méthode est abstract et donc la classe aussi

**ANNEXES : extraits de la documentation Javadoc**

**java.util**

## Class Scanner

|  |  |
| --- | --- |
| **Method Summary** | |
| [String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) | [**next**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#next())()  Finds and returns the next complete token from this scanner. |
| **float** | [**nextFloat**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextFloat())()  Scans the next token of the input as a float. |
| **int** | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextInt())()  Scans the next token of the input as an int. |
| [String](http://docs.oracle.com/javase/6/docs/api/java/lang/String.html) | [**nextLine**](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html#nextLine())()  Advances this scanner past the current line and returns the input that was skipped. |

**Class ArrayList**

|  |  |  |
| --- | --- | --- |
| **Constructor Summary** | | |
| [**ArrayList**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#ArrayList%28%29)()            Constructs an empty list with an initial capacity of ten. | |  |
| [**ArrayList**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#ArrayList%28java.util.Collection%29)([Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<? extends [E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html)> c)            Constructs a list containing the elements (objects) of the specified collection, in the order they are returned by the collection's iterator. | |  |
| **Method Summary** | | |
| boolean | [**add**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#add%28E%29)([E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html) e)            Appends the specified element (object) to the end of this list. | |
| void | [**add**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#add%28int,%20E%29)(int index, [E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html) element)            Inserts the specified element (object) at the specified position in this list. | |
| [E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html) | [**get**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#get%28int%29)(int index)            Returns the element (object) at the specified position in this list. | |
| boolean | [**isEmpty**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#isEmpty%28%29)()            Returns true if this list contains no elements (objects). | |
| [E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html) | [**set**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#set%28int,%20E%29)(int index, [E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html) element)            Replaces the element at the specified position in this list with the specified element (object). | |
| int | [**size**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#size%28%29)()            Returns the number of elements (objects) in this list. | |

**Class Random**

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Random**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#Random())()  Creates a new random number generator. |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| int | [**nextInt**](http://docs.oracle.com/javase/6/docs/api/java/util/Random.html#nextInt(int))(int n)  Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. |

## java.lang

## Class Integer

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Integer**](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#Integer%28int%29)(int value)            Constructs a newly allocated Integer object that represents the specified int value. |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| int | [**intValue**](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#intValue%28%29)**()            Returns the value of this Integer as an int.** |
| static [Integer](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html" \o "class in java.lang) | [**valueOf**](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#valueOf%28int%29)**(int i)            Returns a Integer instance representing the specified int value.** |